Stack with Push, pop and display operations

```
#include<iostream.h>
#include<conio.h>
enum Boolean{FALSE,TRUE};
class stack
{
        int a[10];
        int top;
        int maxsize;
 public:stack()
        {
                top=-1;
                maxsize=10;
        }
        stack(int max)
        {
                top=-1;
                maxsize=max;
        }
        Boolean Isfull();
        Boolean Isempty();
        void push(int);
        void pop();
        void top1();
        void disp();
};

Boolean stack::Isfull()
{
        if(top==maxsize-1)
                return TRUE;
        return FALSE;
}
Boolean stack::Isempty()
{
        if(top==-1)
                return TRUE;
        return FALSE;
}
void stack::push(int x)
{
        if(Isfull())
                cout<<"\nstack is full";
        else a[++top]=x;
```

```
}
void stack::pop()
{
	if(Isempty())
		cout<<"\nstack is empty";
	else cout<<"\npopped element is: "<<a[top--];
}

void stack::top1()
{
	if(Isempty())
		cout<<"\nstack is empty";
	else cout<<"\ntop element is: "<<a[top];
}

void stack::disp()
{
	if(Isempty())
		cout<<"\nstack is empty";
	else
	{	for(int i=top;i>=0;i--)
		{
			cout<<a[i]<<"\t";
		}
	}
}
void main()
{
	stack s;
	int m,ch;

	cout<<"\nenter the max size of the stack:";
	cin>>m;

	s=stack(m);
	cout<<"\n enter the elements for the stack: ";
	do
	{
		cout<<"\n1.push\n2.pop\n3.top\n4. display\n0. exit";
		cout<<"\nenter your choice:";
		cin>>ch;
		switch(ch)
		{
			case 1: cout<<"\nenter element to be pushed";
				int ele;
```

```
                        cin>>ele;
                        s.push(ele);
                        break;
                case 2: s.pop();
                        break;
                case 3: s.top1();
                        break;
                case 4: s.disp();
                        break;
                case 0: break;
                default: break;
            }
        }while(ch!=0);
        getch();
 }
```

Multiple Stacks

```
#include<iostream.h>
#include<conio.h>
class mstack
{
        int a[50], bottom[10], top[10], maxsize, ns;
public: mstack();
        mstack(int,int);
        void push(int,int);
        int pop(int);
        void disp(int);
};
mstack::mstack()
        {
                maxsize=10;
                ns=1;
        }
mstack::mstack(int m,int n)
        {
                maxsize=m;
                ns=n;
                for(int i=0;i<ns;i++)
                        top[i]=bottom[i]=(maxsize/ns)*i-1;
        }
void mstack::push(int ele,int i)
{
        if(top[i]==bottom[i+1]||top[i]==maxsize-1)
                cout<<"\nstack is full";
        else
```

```
                a[++top[i]]=ele;
}
int mstack::pop(int i)
{
        if(top[i]==bottom[i]){
                cout<<"\nstack is empty";
                return -1;
        }
        else return(a[top[i]--]);
}
void mstack::disp(int i)
{
        if(top[i]==bottom[i])
                cout<<"\nstack is empty";
        else
                for(int k=bottom[i]+1;k<=top[i];k++)
                        cout<<a[k]<<"\t";
}
void main()
{
        clrscr();
        mstack ms;int ch,m,n,i,ele;
        cout<<"\nenter the maximum size of the array:";
        cin>>m;
        cout<<"\nenter the no. of stack:";
        cin>>n;
        ms=mstack(m,n);
        do
        {
                cout<<"\n1.insert\n2.delete\n3.display\n0.exit";
                cout<<"\nenter your choice:";
                cin>>ch;
                switch(ch)
                {
                        case 1: cout<<"\nenter the index of the stack:";
                                cin>>i;
                                cout<<"\nenter the element to be inserted:";
                                cin>>ele;
                                ms.push(ele,i);
                                break;
                        case 2: cout<<"\nenter the index of the stack:";
                                cin>>i;
                                cout<<ms.pop(i);
                                break;
                        case 3: cout<<"\nenter the index of the stack:";
```

```
                        cin>>i;
                        ms.disp(i);
                        break;
                case 0: break;
                default: cout<<"\ninvalid input";
            }
        }while(ch!=0);
        getch();
 }
```

Check if a string is palindrome/not using stack:

```
#include<iostream.h>
#include<conio.h>
#include<process.h>
#include<string.h>
#include<stdio.h>
enum Boolean{FALSE,TRUE};
class stack
{       int top;
        char a[20];
        int maxsize;
public: //stack()
//      { maxsize=2;top=-1;}
stack(int size)
{maxsize=size;top=-1;}

        Boolean Isfull()
        {
            if(top==maxsize-1)
                return TRUE;
            return FALSE;
        }

        Boolean Isempty()

        {       if(top==-1)
                    return TRUE;
            return FALSE;
        }
        void push(char x);
        char pop();
        void display();
};
```

```
void stack::push(char x)
{       if(Isfull())
                cout<<"Stack is full \n";
        else
                a[++top]=x;
}

char  stack::pop()
{
        if(Isempty())
        {
                cout<<"stack is empty\n";
                return(-1);
        }
        else
                return(a[top--]);

}
void stack::display()
{       if(Isempty())
                cout<<"stack is empty\n";
        else
        for(int i=top;i>-1;i--)
                cout<<a[i];
}

void main()
{
        stack s(20);
        int option,ele,i;
        char ch,str[20];
        cout<<"Enter a string: ";
        gets(str);

        for(i=0;i<strlen(str);i++)  s.push(str[i]);

        i=0;
        while(str[i]==s.pop() && i<strlen(str) && s.Isempty()==FALSE) i++;
        if(i==strlen(str)-1 && s.Isempty()==TRUE)
                cout<<"string is palindrome";
        else
                cout<<"string is not palindrome";
getch();

}
```

Queue.cpp

```
#include<iostream.h>
#include<conio.h>
enum Boolean{FALSE,TRUE};
class queue
{       int front,rear;
        int a[10];
        int maxsize;
public: queue()
        { maxsize=4;front=rear=-1;}

        Boolean Isfull()
        {
                if(rear==maxsize-1)
                        return TRUE;
                return FALSE;
        }

        Boolean Isempty()
        {       if(rear==front)
                        return TRUE;
                return FALSE;
        }
        void add(int x);
        void del();
        void display();
};

void queue::add(int x)
{       if(Isfull())
                cout<<"Queue is full \n";
         else
                a[++rear]=x;
}

void  queue::del()
{
        if(Isempty())
                cout<<"queue is empty\n";
        else
             cout<<"deleted element is"<<(a[++front]);

}
void queue::display()
```

```
{       if(Isempty())
                cout<<"Queue is empty\n";
        else
        for(int i=front+1;i<=rear;i++)
                cout<<a[i];
}

void main()
{       queue s;
        int option,ele;
        char ch;
        do{
        clrscr();
        cout<<"1.Insert\n2.Delete\n3.display\n4.exit\n";
        cin>>option;
        switch(option)
        {
        case 1:         cout<<"enter the element to be pushed\n";
                        cin>>ele;     s.add(ele);
                        break;
        case 2:         s.del();
                        break;
        case 3:         s.display();break;
        }
        }while(option<4);

      // getch();

}
```

Circular Queue

```
#include<iostream.h>
class queue
{
        int a[10], front, rear, maxsize;
   public:
   queue()
        {
                maxsize=10;
                front=rear=0;
        }
        queue(int m)
        {
                maxsize=m;
```

```
                front=rear=0;
        }
        int Isfull()
        {
                if((rear+1)%(maxsize)==front)
                        return 1;
                else return 0;
        }
        int Isempty()
        {
                if(rear==front)
                        return 1;
                else return 0;
        }
        void insert(int);
        void del();
        void disp();
};
void queue::insert(int e)
{
        if(Isfull())
                cout<<"\nqueue is full";
        else
        {
                rear=(rear+1)%maxsize;
                a[rear]=e;
                cout<<"\nA["<<rear<<"]="<<e;
        }
}
void queue::del()
{
        if(Isempty())
                cout<<"\nqueue is empty";
        else
        {
                front=(front+1)%maxsize;
                cout<<"element deleted is"<<a[front];
        }
}
void queue::disp()
{
    if(Isempty())
        cout<<"Q is empty";
    else
    {
```

```
    for(int i=(front+1)%maxsize;i!=(rear+1)%maxsize;i=(i+1)%maxsize)
        cout<<a[i]<<"\t";
        }
 }
int main (int argc, char *argv[])
 {
        //clrscr();
        queue q;int ch,m,ele;
        cout<<"\nEnter the maximum size of the array:";
        cin>>m;
        q=queue(m);
        do
        {
                cout<<"\n1.Insert\n2.Delete\n3.Display\n0.Exit";
                cout<<"\nenter your choice:";
                cin>>ch;
                switch(ch)
                {
                        case 1: cout<<"\nenter the element to be inserted:";
                                cin>>ele;
                                q.insert(ele);
                                break;
                        case 2: q.del();
                                break;
                        case 3: q.disp();
                                break;
                        case 0: break;
                        default: cout<<"\ninvalid input";
                }
        }while(ch!=0);
        //getch();

        return 0;
 }
```

Template

```
#include<iostream.h>
#include<conio.h>

template<class TA>
void swap(TA &a,TA &b)
{       TA temp=a;
        a=b;
        b=temp;
}
```

```
template<class TA>
void sort(TA a[],int n)
{
	int i,j;
	for(i=0;i<n-1;i++)
		for(j=0;j<n-1-i;j++)
			if(a[j]>a[j+1])
				swap(a[j],a[j+1]);
		for( i=0;i<4;i++)
			cout<<a[i]<<" ";

}

void main()
{
	double a[10];
	clrscr();
	for(int i=0;i<4;i++)
		cin>>a[i];
	sort(a,4);
	getch();
}
```

Prior to postfix conversion

```
#include<iostream.h>
#include<string.h>
#include<stdio.h>
#include<conio.h>

void main()
{
clrscr();
char str[10];
int i=0,number=0;
cout<<"Enter the string: ";
gets(str);

for(i=0;i<strlen(str);i++)
	number=number+(str[i]-'0');

cout<<number;
getch();
}
```

Postfix Evaluation

```
#include<iostream.h>
#include<string.h>
#include<stdio.h>
#include<conio.h>
```

```
const int MAX=100;
class STACK
{
	int TOP;
	int stack[MAX];
	public:
	STACK()
	{
		TOP=-1;
	}
	void push(int);
	int pop();
	int  returntop()
	{
		return TOP;
	}
};
void STACK::push(int opernd)
{
	stack[++TOP]=opernd;
}
int STACK::pop()
{
	if(TOP==-1)
		return -1;
	else
		return stack[TOP--];
}
void postfix_evaluation()
{
	STACK S;
	int oper,op1,op2,c,cop,j;
	char symbol,postfix[50];
	cout<<"ENTER THE POSTFIX EXPRESSION:";
	gets(postfix);
	for(int i=0;i<strlen(postfix);i++)
	{
		symbol=postfix[i];

		if( symbol >= 48 && symbol<= 57)
		{
		  oper=symbol-'0';
		  S.push(oper);
		}

	else if(symbol=='+' || symbol=='-' || symbol=='*' || symbol=='/' || symbol == '%')
		{
			op2=S.pop();
			op1=S.pop();
			switch(symbol)
			{	case '%': c=op1%op2;
```

```
                                        S.push(c);
                                        break;
                                case '/': c=op1/op2;
                                        S.push(c);
                                        break;
                                case '*': c=op1*op2;
                                        S.push(c);
                                        break;
                                case '+': c=op1+op2;
                                        S.push(c);
                                        break;
                                case '-': c=op1-op2;
                                        S.push(c);
                                        break;
                        }
                }
                else
                {
                 int val;
                 cout<<"Enter the value for "<<symbol;
                 cin>>val;
                 S.push(val);
                }
        }//for
        cout<<"the result is = "<<S.pop();
}
void main()
{
        clrscr();
        postfix_evaluation();
        getch();
}
```

Infix to postfix conversion

```
#include<iostream.h>
#include<conio.h>

enum Boolean{FALSE,TRUE};

int icp[]={20,19,12,12,13,13,13};
int isp[]={0,19,12,12,13,13,13};
enum precedence{lparen,rparen,plus,minus,times,divide,mod,operand};

class stack
{       int top;
        char a[10];
        int maxsize;
public: stack()
        { maxsize=10;top=-1;}

        Boolean Isfull()
        {
```

```
                if(top==maxsize-1)
                        return TRUE;
                return FALSE;
        }

        Boolean Isempty()

        {       if(top==-1)
                        return TRUE;
                return FALSE;
        }
        void push(char x);
        char  pop();
        void display();
        char topele();
};
void stack::push(char x)
{       if(Isfull())
                cout<<"Stack is full \n";
        else
                a[++top]=x;
}
char stack::topele()
{       return(a[top]);
}
char  stack::pop()
{
        if(Isempty())
                return(-9999);
        else
                  return(a[top--]);

}
void  stack::display()
{       if(Isempty())
                cout<<"Stack is empty";
        else
        for(int i=top;i>-1;i--)
                cout<<a[i];
}


precedence get_token(char c)
{
        switch(c)
        {
        case '(':return lparen;
        case ')':return rparen;
        case '+':return plus;
        case '-':return minus;
        case '*':return times;
        case '/':return divide;
```

```
        case '%':return mod;
        default:return operand;
        }
}
void postfix(char infix[])
{
        precedence temp;
        int i=0;
        stack s;
        while(infix[i]!='\0')
        {
                temp=get_token(infix[i]);
                if(temp==operand)
                        {cout<<infix[i];}
                else if(temp==rparen)
                        {
                        while(get_token(s.topele())!=lparen)
                                cout<<s.pop();
                        char c=s.pop();
                         }
                else
                {
                        if(s.Isempty()==TRUE)
                                s.push(infix[i]);
                          else
                          {

                        while(icp[temp]<=isp[get_token(s.topele())]&&s.Isempty()==FALSE)
                                        cout<<s.pop();
                                s.push(infix[i]);
                           }
                }

                i++;
        }
        s.display();
        }
        void main()
        { char infix[10];
                cout<<"enter infix expression\n";
                cin>>infix;
                postfix(infix);
                getch();
        }
```

Infix to prefix Conversion

```
#include<iostream.h>
#include<string.h>
#include<conio.h>

enum Boolean{FALSE,TRUE};
```

```
int icp[]={20,19,12,12,13,13,13,0};
int isp[]={0,19,12,12,13,13,13,0};
enum precedence{lparen,rparen,plus,minus,times,divide,mod,eos,operand};

class stack
{       int top;
        char a[10];
        int maxsize;
public: stack()
        { maxsize=10;top=-1;}

        Boolean Isfull()
        {
                if(top==maxsize-1)
                        return TRUE;
                return FALSE;
        }

        Boolean Isempty()

        {       if(top==-1)
                        return TRUE;
                return FALSE;
        }
        void push(char x);
        char  pop();
        void display();
        char topele();
};
void stack::push(char x)
{     if(Isfull())
                cout<<"Stack is full \n";
        else
                a[++top]=x;
}
char stack::topele()
{       return(a[top]);
}
char  stack::pop()
{
        if(Isempty())
                return(-9999);
        else
                  return(a[top--]);

}
void  stack::display()
{      if(Isempty())
                cout<<"Stack is empty";
        else
        for(int i=top;i>-1;i--)
                cout<<a[i];
```

```
}


precedence get_token(char c)
{
        switch(c)
        {
        case '(':return lparen;
        case ')':return rparen;
        case '+':return plus;
        case '-':return minus;
        case '*':return times;
        case '/':return divide;
        case '%':return mod;
        default:return operand;
        }
}
void postfix(char infix[])
{
        precedence temp;
        char prefix[20];
        int i=0,p=0;
        stack s;

        while(infix[i]!='\0')
        {
                temp=get_token(infix[i]);

                if(temp==operand)
                        prefix[p++]=infix[i];
                else if(temp==rparen)
                        {//char c1=s.topele();

                        while(get_token(s.topele())!=lparen)
                                prefix[p++]=s.pop();
                        char c=s.pop();
                         }

                else
                {
                        if(s.Isempty()==TRUE)
                                s.push(infix[i]);
                          else
                          {

while(icp[temp]<isp[get_token(s.topele())]&&s.Isempty()==FALSE)
                                                prefix[p++]=s.pop();
                                s.push(infix[i]);
                           }
                }
```

```
            i++;
        }
        while(!s.Isempty())
            prefix[p++]=s.pop();
        prefix[p]='\0';

        cout<<strrev(prefix);
}
        void main()
        {
            char infix[20],temp[20];
            int j=0;
            cout<<"enter infix expression\n";
            cin>>infix;
            for(int i=strlen(infix)-1;i>=0;i--)
            {
                if(infix[i]=='(')       temp[j++]=')';
                else if(infix[i]==')')  temp[j++]='(';
                else                    temp[j++]=infix[i];
            }
            temp[j]='\0';

            postfix(temp);
            getch();
        }
```

Prefix Evalution

```
#include<iostream.h>
#include<string.h>
#include<stdio.h>
#include<conio.h>
const int MAX=100;
class STACK
{
        int TOP;
        int stack[MAX];
        public:
        STACK()
        {
            TOP=-1;
        }
        void push(int opernd)
        {
        stack[++TOP]=opernd;
        }
        int pop()
        {
            int a;
            if(TOP==-1)
            {
                return -1;
            }
```

```
                else
                {
                        return stack[TOP--];
                }
        }
        int  returntop()
        {
                return TOP;
        }
};
class EVALUATION
{
        char prefix[20];
        char symbol;
        public:
        void prefix_evaluation()
        {
                STACK S;
                int oper,a[2],c,cop,j;
                int len;
                char symbol;
                cout<<"ENTER THE PREFIX EXPRESSION:";
                gets(prefix);
                len=strlen(prefix);

                for(int i=len-1;i>=0;i--)
                {
                        symbol=prefix[i];
                        if( symbol >= 48 && symbol<= 57)
                                S.push(symbol-'0');
                        else
                        {
                                for(j=0;j<2;j++)
                                {
                                        a[j]=S.pop();
                                }

                                switch(symbol)
                                {
                                        case '%': c=a[0]%a[1];
                                                S.push(c);
                                                break;
                                        case '/': c=a[0]/a[1];
                                                S.push(c);
                                                break;
                                        case '*': c=a[0]*a[1];
                                                S.push(c);
                                                break;
                                        case '+': c=a[0]+a[1];
                                                S.push(c);
                                                break;
                                        case '-': c=a[0]-a[1];
```

```
                                        S.push(c);
                                        break;
                                }
                        }
                }
                cout<<"RESULT : "<<S.pop();
        }
};
void main()
{
        EVALUATION E;
        clrscr();
        E.prefix_evaluation();
        getch();
}
```

Postfix to fully parenthesized Infix

```
#include<iostream.h>
#include<string.h>
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
const int MAX=50;
class STACK
{
        int TOP;
        char stack[MAX][MAX];
        public:
        STACK()
        {
                TOP=-1;
        }
        void push(char []);
        char* pop();
};
void STACK::push(char opernd[])
{       TOP++;
          strcpy(stack[TOP],opernd);
}
char* STACK::pop()
{
        if(TOP==-1)
                return "X";
        else
                return stack[TOP--];
}

void postfix_infix()
{
        STACK s;
        int c,cop,j;
        char oper[20],op1[20],op2[20];
        char symbol[20],postfix[20];
```

```
	cout<<"ENTER THE POSTFIX EXPRESSION:";
	gets(postfix);

	for(int i=0;i<strlen(postfix);i++)
	{
		char temp1[20],temp2[20],temp3[20];
		temp1[0]=postfix[i];
		temp1[1]='\0';
		strcpy(symbol,temp1);

		if(!isalpha(symbol[0]))
		{
			strcpy(op2,s.pop());
			strcpy(op1,s.pop());
			strcpy(temp3,"(");
			strcat(temp3,op1);
			strcat(temp3,symbol);
			strcat(temp3,op2);
			strcat(temp3,")");
			s.push(temp3);
		}
		else s.push(temp1);
	}
	cout<<"the result is = "<<s.pop();
}
void main()
{
	clrscr();
	postfix_infix();
	getch();
}
```

Prefix to fully parenthesized Infix

Sparse Matrix

```
#include<iostream.h>
#include<conio.h>

class sparse
{
	int row;
	int col;
	int value;
public:
	sparse(int r,int c,int v)
	{
		row=r;
		col=c;
		value=v;
```

```
	}
	sparse()
	{
		row=col=value=0;
	}
	void disp(sparse*);
};

void sparse::disp(sparse a[])
{
	cout<<"row\tcol\tvalue\n";
	for(int i=0;i<=a[0].value;i++)
	{
cout<<a[i].row<<"\t"<<a[i].col<<"\t"<<a[i].value<<"\n";
	}
}
int main (int argc, char *argv[])
{
	sparse s[20],d;		int r,c,v,val;
	cout<<"Enter the no. of rows:\n";	cin>>r;
	cout<<"Enter the no. of cols:\n";	cin>>c;
	cout<<"Enter the no. of values:\n";	cin>>v;
	s[0]=sparse(r,c,v);

cout<<"Enter the elements of the sparse matrix:\n";
	for(int i=1;i<=v;i++)
	{
	cout<<"\nrow "<<i<<": ";	cin>>r;
	cout<<"\ncol "<<i<<": ";	cin>>c;
	cout<<"\nvalue "<<i<<": ";	cin>>val;
		s[i]=sparse(r,c,val);
	}
	d.disp(s);
}
```

Slow Transpose

```
#include<iostream.h>
#include<conio.h>

class sparse
{
	int row;
	int col;
	int value;
public:
sparse(int r,int c,int v)
	{
		row=r;
		col=c;
		value=v;
	}
```

```
        sparse()
        {
                row=col=value=0;
        }
        void slowTranspose(sparse*);
        void disp(sparse*);
};
void sparse::slowTranspose(sparse a[])
{
        sparse b[20];
        int ctr=1;

for(int i=0;i<a[0].col;i++)
{
   for(int j=1;j<=a[0].value;j++)
        {
                        if(a[j].col==i)
                        {
                                b[ctr].row=a[j].col;
                                b[ctr].col=a[j].row;
                                b[ctr].value=a[j].value;
                                ctr++;
                        }
   }
}
        b[0].col=a[0].row;
        b[0].row=a[0].col;
        b[0].value=a[0].value;
        cout<<"INPUT SPARSE MATRIX IS:\n";
        disp(a);
        cout<<"OUTPUT USING SLOW TRANSPOSE:\n";
        disp(b);
}
void sparse::disp(sparse a[])
{
        cout<<"row\tcol\tvalue\n";
        for(int i=0;i<=a[0].value;i++)
        {
                cout<<a[i].row<<"\t"<<a[i].col<<"\t"<<a[i].value<<"\n";
        }
}

void main(int argc, char *argv[])
{
        sparse s[20];
        int r,c,v,val;
        cout<<"Enter the no. of rows:\n";
        cin>>r;
        cout<<"Enter the no. of cols:\n";
        cin>>c;
        cout<<"Enter the no. of values:\n";
        cin>>v;
```

```
	s[0]=sparse(r,c,v);
	cout<<"Enter the elements of the sparse matrix:\n";
	for(int i=1;i<=v;i++)
	{
		cout<<"\nrow "<<i<<": ";
		cin>>r;
		cout<<"\ncol "<<i<<": ";
		cin>>c;
		cout<<"\nvalue "<<i<<": ";
		cin>>val;
		s[i]=sparse(r,c,val);
	}
	s[0].slowTranspose(s);
	getch();
}
```

Fast Transpose

```
#include<iostream.h>
#include<conio.h>
class sparse
{
	int row,col,value;
 public:
sparse(int r,int c,int v)
	{
		row=r;
		col=c;
		value=v;
	}
	sparse()
	{
		row=col=value=0;
	}
	void fastTranspose(sparse*);
	void disp(sparse*);
};

void sparse::fastTranspose(sparse a[])
{
	sparse b[20];
	int rowterm[20], start_pos[20], i, j, k;
	b[0].col=a[0].row;
	b[0].row=a[0].col;
	b[0].value=a[0].value;
// place 0s into rowterm array. Before this the rowterm has all random values
	for(i=0;i<a[0].col;i++)
		rowterm[i]=0;
	for(j=1;j<=a[0].value;j++)
		rowterm[a[j].col]++;
	start_pos[0]=1;
	for(k=1;k<a[0].col;k++)
		start_pos[k]=start_pos[k-1]+rowterm[k-1];
```

```
	for(k=1;k<=a[0].value;k++)
	{
		int x=start_pos[a[k].col];
		b[x].row=a[k].col;
		b[x].col=a[k].row;
		b[x].value=a[k].value;
		start_pos[a[k].col]++;
	}
	cout<<"OUTPUT USING FAST TRANSPOSE:\n";
	disp(b);
}
void sparse::disp(sparse a[])
{
	cout<<"row\tcol\tvalue\n";
	for(int i=0;i<=a[0].value;i++)
	{
		cout<<a[i].row<<"\t"<<a[i].col<<"\t"<<a[i].value<<"\n";
	}
}

int main (int argc, char *argv[])
{
	clrscr();
	sparse s[20];
	int r,c,v,val;
	cout<<"Enter the no. of rows:\n";	cin>>r;
	cout<<"Enter the no. of cols:\n";	cin>>c;
	cout<<"Enter the no. of values:\n";	cin>>v;
	s[0]=sparse(r,c,v);

	cout<<"Enter the elements of the sparse matrix:\n";
	for(int i=1;i<=v;i++)
	{
		cout<<"\nrow "<<i<<": ";		cin>>r;
		cout<<"\ncol "<<i<<": ";		cin>>c;
		cout<<"\nvalue "<<i<<": ";		cin>>val;
		s[i]=sparse(r,c,val);
	}
	s[0].fastTranspose(s);
	getch();
}
```

Pointers

P1.cpp

```
#include<iostream.h>
#include<conio.h>
void main()
{
	clrscr();
	int i=3;
```

```
	cout<<"i: "<<i<<endl;		getch();
	cout<<"&i: "<<&i<<endl;	getch();
}
```

P2.cpp

```
#include<iostream.h>
#include<conio.h>
void main()
{
	clrscr();
	int i=3;
	cout<<"i: "<<i<<endl;		getch();
	cout<<"&i: "<<&i<<endl;			getch();
	cout<<"*(&i): "<<*(&i);		getch();
}
```

P3.cpp

```
#include<iostream.h>
#include<conio.h>
void main()
{
	clrscr();
	int i=3, *j=i;
     //	j=&i;
 //
	cout<<"&i: "<<&i<<endl;			getch();
	cout<<"j: "<<j<<endl<<endl;		getch();

	cout<<"&j: "<<&j<<endl;	getch();

	//cout<<"j: "<<j<<endl;		getch();
	cout<<"i: "<<i<<endl;			getch();
	cout<<"*(&i): "<<*(&i)<<endl;	getch();
	cout<<"*j: "<<*j;			getch();
}
```

P4.cpp

```
#include<iostream.h>
#include<conio.h>
void main()
{
	clrscr();
	char c, *cc;
	int i, *ii;
	float f, *ff;
	c='A';		cc=&c;
	i=54;	ii=&i;
	f=3.14;		ff=&f;
	cout<<"cc: "<<(int *)&c<<endl;				getch();
	cout<<"*cc: "<<*cc<<endl<<endl;				getch();

	cout<<"ii: "<<ii<<endl;				getch();
	cout<<"*ii: "<<*ii<<endl<<endl;				getch();
```

```
	cout<<"ff: "<<ff<<endl;				getch();
	cout<<"*ff: "<<*ff<<endl<<endl;				getch();

}
```

P5.cpp

```
#include<iostream.h>
#include<conio.h>
void main()
{	clrscr();
	char r, *cc;
	int i, *ii;
	float f, *ff;

	r='A';		cc=&r;
	i=54;	ii=&i;
	f=3.14;		ff=&f;

	cout<<"cc: "<<(int *)cc<<"\t cc+1: "<<(int *)cc+1<<endl;  getch();
	cout<<"ii: "<<ii<<"\t ii+1: "<<ii+1<<endl;   getch();
	cout<<"ff: "<<ff<<"\t ff+1 "<<ff+1<<endl;   getch();
}
```

P6.cpp

```
#include<iostream.h>
#include<conio.h>
void disp(int x1,int y1)
{	cout<<"x= "<<x1<<" y= "<<y1<<endl;	}

void swap(int a,int b)
{	int t=a;
	a=b;
	b=t;
}
void main()
{	clrscr();
	int x=10, y=20;
	cout<<"Before swapping: ";   disp(x,y);
	swap(x,y);
	cout<<"After swapping: ";     disp(x,y);
	getch();
}
```

P7.cpp

```
#include<iostream.h>
#include<conio.h>

void main()
{
	clrscr();
	int a[]={1,2,3,4,5};int i;
	cout<<"a: (base address) "<<a<<endl<<endl;			getch();
	for(i=0;i<5;i++)
	{
		cout<<"a["<<i<<"]="<<a[i]<<endl;   getch();
```

```
		cout<<"a+"<<i<<"="<<a+i<<endl;	getch();
		cout<<"*(a+"<<i<<")="<<*(a+i)<<endl;	getch();
		cout<<endl;
	}}
```

P8.cpp (Pointer Array)

```
#include<iostream.h>
#include<conio.h>
void main()
{
	clrscr();
	int a[3][4]={{10,20,30,40},{11,21,31,41},{12,22,32,42}};
	int m=3, n=4, i, j;
	int *p[4];

	for(i=0;i<m;i++)
		p[i]=&a[i][0];


	for(i=0;i<m;i++)
		for(j=0;j<n;j++)
			cout<<"\nAddess: "<<p[i]+j<<" content: "<<*(p[i]+j);

	getch();
}
```

P9.cpp (String Copy)

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
void main()
{
	char src[20], dest[20];
	char *ptr1=src, *ptr2=dest;
	clrscr();
	cout<<"Enter the String: ";	gets(src);
/*
	while(*ptr1!='\0')
	{
		*ptr2=*ptr1;
		ptr1++;
		ptr2++;
	}	*/

//	while(*ptr1)
  //		*ptr2++=*ptr1++;

	while(*ptr2++=*ptr1++);
	*ptr2='\0';
	cout<<"Result= "<<dest;
	getch();
}
```

P10.cpp (2D array Pointer)

```
#include<iostream.h>
```

```
#include<conio.h>
void main()
{
	clrscr();
	int a[3][4]={{10,20,30,40},{11,21,31,41},{12,22,32,42}};
	int m=3, n=4, i, j;
	int (*p)[4];
	p=a;
/*
	for(i=0;i<m;i++)
		for(j=0;j<n;j++)
			cin>>*(*(p+i)+j);*/

	for(i=0;i<m;i++)
		for(j=0;j<n;j++)
			cout<<*(*(p+i)+j);
	getch();
}
```

P11.cpp (Pointer to function 1)

```
#include<iostream.h>
#include<conio.h>
void func (int a, int b)
{
   cout<<"\n a= "<<a;
  cout<<"\n b = "<<b;
}
void main()
{
   void (*fptr)(int,int); // Function pointer
   fptr = func; // Assign address to function pointer
clrscr();
   func(2,3);
   fptr(2,3);
getch();
}
```

P12.cpp (pointer to function2)

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
#include<stdio.h>
#include<string.h>
void disp (double (*fun)(double), int  n)
{
  for (int i=0; i<=n; i++)
	cout<<"\n"<<i<<"    "<<(*fun) (i);

}
void main()
{
 int n;
char str[10];
double (*ptr)(double);
```

```
clrscr();
cout<<" Enter 'sin' for sine series and 'cos' for cosine series: ";
gets(str);
if(strcmp(str,"sin")==0)
        ptr=sin;
else
        ptr=cos;
disp(ptr,5);
getch();
}
```

	Function \Check	**Column 1 if(first==NULL)**	**Column 2**	**Column 3**
Insert a new node to the end of the list	create()/insEnd()	first=temp		
Insert a new node to the begining of the list	insFront()	first=temp	else {temp->next=first; first=temp;}	
Insert a node at the *N*th position in the list	insN()	if(first==NULL&&n==1) then first=temp		else if(n<count) *//count number of nodes in the list* { for(list *curr=first;i<n;curr=curr->link,i+- temp->link=curr->link; curr->link=temp; }
Insert a new node at before a node containing the value x	insBefore (int x)	Display “list is empty” and return	If(x==first->data) { temp->next=first; first=temp; flag=1; }	else{ for(list *curr=first;curr!=NULL;prev=curr { if(curr->data==x) { temp->next=curr; prev->next=temp; flag=1; break; } }
Insert a new node at after a node containing the value x	insAfter(int x)	Display “list is empty” and return		for(list *curr=first;curr!=NULL;curr=curr if(curr->data==x) { temp->next=curr->next curr->next=temp; flag=1; break; } }

Delete the first node of the list	delFront( )	Display “list is empty” and return	else{ list *temp=first; first=first->link; delete temp; }	
Delete the last node of the list	delLast()	Display “list is empty” and return	if(first>link==NULL) { list *temp=first; first=NULL; delete temp; }	else{ list *prev; for(list *curr=first;curr->link!=NULL; pre >link) prev->link=NULL; delete curr; }
Delete the node containing the value x	del_speci fic(int x)	Display “list is empty” and return	if(first->data==x){ list *temp=first; first=first->link; delete temp; }	else{ list *prev;int flag=0; for(list *curr=first;curr!=NULL;prev=curr if(curr->data==x) { prev->link=curr delete curr; flag=1; break; } }

Linked List

```
#include<iostream.h>
#include<conio.h>

class list
{
        int data;
        list *link;
 public:
        void ins_end();
        void ins_front();
        void ins_n(int n);

        void delSpecific();
        void del_front();
        void del_end();
        int count();
        void sort();
        void traverse();
        void reverse();
};
list *first=NULL;

 void list::reverse()
```

```
{
        list *curr,*prev;
        curr=prev=NULL;
        while(first!=NULL)
        {
                prev=curr;
                curr=first;
                first=first->link;
                curr->link=prev;
        }
        first=curr;
}



void list::ins_n(int n)
{
        int count=0,i=1;
        for(list *curr=first;curr!=NULL;curr=curr->link)      count++;
        cout<<"Count= "<<count;
        list *temp=new list;
        cout<<" Enter data: ";
        cin>>temp->data;
        temp->link=NULL;

        if(n==1&&count==0) first=temp;
        else if(n<count)
        {
                for(list *curr=first;i<n;curr=curr->link,i++);
                temp->link=curr->link;
                curr->link=temp;
        }
        traverse();
}

int list::count()
{
        list *p;
        int count=0;
        for(p=first;p!=NULL;p=p->link)
                count++;
        return count;
}

void list::ins_end()
{
        list *temp;
        temp=new list;
        cout<<"\nenter the data:";
        cin>>temp->data;
        temp->link=NULL;
```

```
	if(first==NULL)
		first=temp;
	else
	{
		for(list *curr=first;curr->link!=NULL;curr=curr->link);
		curr->link=temp;
	}
	traverse();
}

void list::ins_front()
{
	list *temp;
	temp=new list;
	cout<<"\nenter the data:";
	cin>>temp->data;
	temp->link=NULL;

	if(first==NULL)
		first=temp;
	else
	{
		temp->link=first;
		first=temp;
	}
	traverse();
}

void list::delSpecific()
{
	int x;
	cout<<"\nenter the element to be deleted:";
	cin>>x;

	if(first==NULL)
		cout<<"\nlist is empty";
	else if(first->data==x)
		{
			list *temp=first;
			first=first->link;
			delete temp;
		}
	else
		{	list *prev;
			int flag=0;
			for(list *curr=first;curr!=NULL;curr=curr->link)
			{
				if(curr->data==x)
				{
					prev->link=curr->link;
					delete curr;
					flag=1;
```

```
                                        break;
                                }
                                prev=curr;
                        }
                        if(flag==0)     cout<<"\nnode not found";
                }
                traverse();
}

void list::del_front()
{
        if(first==NULL)
                cout<<"\nlist is empty";
        else
        {
                list *temp=first;
                first=first->link;
                delete temp;
        }
                traverse();
}

void list::del_end()
{
        if(first==NULL)
                cout<<"\nlist is empty";
        else if(first->link==NULL)
        {
                list *temp=first;
                first=NULL;
                delete temp;
        }
        else
        {
                list *prev;
                for(list *curr=first;curr->link!=NULL;curr=curr->link)
                                prev=curr;
                prev->link=NULL;
                delete curr;
        }
                traverse();
}

void list::sort()
{
        for(list *i=first;i->link!=NULL;i=i->link)
        {
                for(list *j=i->link;j!=NULL;j=j->link)
                {
                        if(i->data>j->data)
                        {
                                int d=i->data;
```

```
                        i->data=j->data;
                        j->data=d;
                    }
                }
        }
        cout<<"\nthe sorted list is";
        traverse();
}



void list::traverse()
{
        if(first==NULL)
                cout<<"\nlist is empty";
        else
        {
                cout<<"\nthe list is:\n";
                for(list *curr=first;curr!=NULL;curr=curr->link)
                        cout<<curr->data<<"\t";
        }
}


void main()
{
        clrscr();
        list l;
        int ch,n;
        do
        {
                cout<<"\n1 insert end\t2 insert front\t\t3. delete front\n4 delete end\t5. delete
specific\t6.traverse\t7.count\n0.exit";
                cout<<" 8. Insert nth node \n 9. sort 10. reverse \nenter your choice:";
                cin>>ch;
                switch(ch)
                {
                        case 1:l.ins_end();break;
                        case 2:l.ins_front();break;

                        case 3:l.del_front();break;
                        case 4:l.del_end();break;
                        case 5:l.delSpecific();break;

                        case 6:l.traverse();break;
                        case 7:cout<<"\nNo of nodes: "<<l.count()<<endl; break;
                        case 8: cout<<"Enter value for n: ";cin>>n; l.ins_n(n);break;
                        case 9:l.sort(); break;
                        case 10: l.reverse();break;
                        case 0:break;
                        default:cout<<"\ninvalid input";break;
```

```
                }
        }while(ch!=0);
        getch();
}
```

Concatenate and merge two lists

```
#include <iostream>
using namespace std;
class list
{
        int data;
        list *link;
 public:
        list* create(list*,int);
        void traverse(list *);
        list* merge(list *firsta,list *firstb);
void mwithout_new(list *,list *);
};

list *first1=NULL,*first2=NULL,*first3=NULL;

list* list::create(list *first, int ele)
{
        list *temp,*curr;
        temp=new list;
        temp->data=ele;
        temp->link=NULL;
        if(first==NULL)
                first=temp;
        else
        {
                for(curr=first;curr->link!=NULL;curr=curr->link);
                curr->link=temp;
        }
        return first;
}
void list::traverse(list *first)
{
        if(first==NULL)
                cout<<"\nlist is empty";
        else
        {
                cout<<"\nthe list is:\n";
                for(list *curr=first;curr!=NULL;curr=curr->link)
                        cout<<curr->data<<"\t";
        }
}

list* list::merge(list *firsta,list *firstb)
{
        list *curra=firsta;
        list *prev=NULL;
```

```
	list *currb=firstb;
	list *res=NULL;

	while(curra!=NULL&&currb!=NULL)
	{
		if(curra->data<=currb->data)
		{
			res=create(res,curra->data);
			curra=curra->link;
		}
		else
		{
			res=create(res,currb->data);
			currb=currb->link;
		}
	}
	while(curra!=NULL)
	{
		res=create(res,curra->data);
		curra=curra->link;
	}
	while(currb!=NULL)
	{
		res=create(res,currb->data);
		currb=currb->link;
	}
	traverse(res);
	return(res);
}

void list::mwithout_new(list *a, list *b){
list *currA = a, *currB = b, *currALink, *currBLink;
while(currA != NULL && currB != NULL){
while (currA->data < currB->data && currA->link != NULL)
if(currA->link->data < currB->data)
currA = currA->link;
else
break;

if(currB->data > currA->data){
currALink = currA->link;
currBLink = currB->link;
currA->link = currB;
currB->link = currALink;
currB = currBLink;
}
}
}
int main()
{
//	clrscr();
	list l1,l2;
```

```
	int ch,ele;
	do
	{
		cout<<"\n1 create X1\n2 traverse X1\n3 create X2\n4. traverse X2\n5. new
merge X1andX2\n 6. Merge \n0. exit\n";
		cout<<"\nenter your choice:";
		cin>>ch;
		switch(ch)
		{
			case 1: cout<<"\nenter the data:";
				cin>>ele;
				first1=l1.create(first1,ele);
				break;
			case 2:l1.traverse(first1);break;
			case 3: cout<<"\nenter the data:";
				cin>>ele;
				first2=l1.create(first2,ele);
				break;
			case 4:l1.traverse(first2);break;
			case 5:cout<<"THE CONCATENATED LIST IS:";
				l1.mwithout_new(first1,first2);break;
			case 6:
				first3=l1.merge(first1,first2);break;
		  //	default:cout<<"\ninvalid input";break;
		}
	}while(ch!=0);
//	getch();
return 0;
}



void list::concatenate(list *f1,list *f2)
{
	if(f1==NULL)
	{
		f1=f2;
	}
	else if(f2==NULL)
	{}
	else
	{
		for(list*curr=f1;curr->link!=NULL;curr=curr->link);
		curr->link=f2;
	}
	traverse(f1);
}
```

Circular List

```
#include<iostream.h>
#include<conio.h>
#include<process.h>
```

```
class clist
{
	int data;
	clist *next;
public:
	//clist();
	void insert();
	void dele();
	void disp();
	void init();
};
clist *headnode;

void clist::init()
{
	headnode=new clist;
	headnode->next=headnode;
}

void clist::insert()
{
	clist *temp,*curr;
	temp=new clist;
	cout<<"enter the data\n";
	cin>>temp->data;
	curr=headnode->next;

	if(headnode->next==headnode)
	{
	 temp->next=headnode;
	 headnode->next=temp;
	}

	else{
		while(curr->next!=headnode)
			curr=curr->next;

		temp->next=curr->next;
		curr->next=temp;
	}
}
void clist::dele()
{
	clist *curr=headnode->next,*prev;
	if(headnode->next==headnode)
		cout<<"List is empty\n";
	else{
		if(curr->next==headnode)
		{
			headnode->next=headnode;
			delete(curr);
```

```
                }
                else{
                        while(curr->next!=headnode)
                        {
                                prev=curr;
                                curr=curr->next;
                        }
                        prev->next=headnode;
                        delete(curr);
                }
                disp();
        }
}

 void clist::disp()
 {
        clist *curr=headnode->next;
        if(headnode->next==headnode)
                cout<<"empty list";
        else
        {
                while(curr!=headnode)
                {
                        cout<<"Address: "<<curr<<"\t data: "<<curr->data<<endl;
                        getch();
                        curr=curr->next;
                }
        }
 }

 void main()
 {
        clrscr();
        clist a;
        int ch;
        a.init();

        do
        {
                cout<<"\n1.insert\n2.delete\n3.disp\t\n4.exit\nEnter your choice: ";
                cin>>ch;
                switch (ch)
                {
                        case 1: a.insert();       break;
                        case 2: a.dele();         break;
                        case 3: a.disp();      break;
                        case 4: exit(0);
                }
        }while(1);
        //getch();
}
```

Polynomial Addition and Multiplication

```
#include<iostream.h>
#include<conio.h>

class node
{
        int exp;
        int coe;
        node *next;

   public:
        node *create(node *,int,int);
        void display(node *);
        node *add(node *,node *);
        node *mul(node *,node *);
};
node *first1=NULL,*first2=NULL,*first3=NULL,*first4=NULL;

node * node::mul(node *f1,node *f2)
{
        node *res=NULL;
        for(node *i=f1;i!=NULL;i=i->next)
                for(node *j=f2;j!=NULL;j=j->next)
                        res=create(res,i->coe*j->coe,i->exp+j->exp);
        return(res);
}



node * node::create(node *f,int c,int e)
{
                node *temp;
        temp=new node;

        temp->coe=c;
        temp->exp=e;
        temp->next=NULL;

        if(f==NULL)  f=temp;
        else
        {
                node *curr;
                for(curr=f;curr->next!=NULL;curr=curr->next);
                curr->next=temp;
        }
        return(f);
 }


   void node::display(node *f)
   {
```

```
        for(node *curr=f;curr!=NULL;curr=curr->next)
        {
           cout<<curr->coe<<"^"<<curr->exp;
           if(curr->next!=NULL)
             cout<<"+";
        }
        cout<<"\n";
    }

node* node::add(node *f1,node *f2)
{
        node *res=NULL;
        node *a=f1,*b=f2;

        while((a!=NULL)&&(b!=NULL))
        {
                if (a->exp>b->exp)
                {
                        res=create(res,a->coe,a->exp);
                        a=a->next;
                }
                else if(a->exp==b->exp)
                {
                        res=create(res,a->coe+b->coe,a->exp);
                        a=a->next;
                        b=b->next;
                }
                else //b->exp<a->exp
                {
                        res=create(res,b->coe,b->exp);
                        b=b->next;
                }
        }
        while(a!=NULL)
        {
                res=create(res,a->coe,a->exp);
                a=a->next;
        }
        while(b!=NULL)
        {
                res=create(res,b->coe,b->exp);
                b=b->next;
        }
        return(res);
}

int main()
{
        int n,i,e,c;
        node a;
        clrscr();
```

```
	cout<<"how many terms in 1 polynomial\n";
	cin>>n;
	for(i=0;i<n;i++)
	{
		cout<<"\nEnter coe: ";
		cin>>c;
		cout<<"Enter exp: ";   cin>>e;
		first1=a.create(first1,c,e);
	}
	a.display(first1);

	cout<<"how many terms in 2 polynomial\n";
	cin>>n;

	for(i=0;i<n;i++)
	{
		cout<<"\nEnter coe: ";
		cin>>c;
		cout<<"Enter exp: ";   cin>>e;
		first2=a.create(first2,c,e);
	}
	a.display(first2);
	first3=a.add(first1,first2);
	first4=a.mul(first1,first2);
	cout<<"Result of addition: ";		a.display(first3);	cout<<endl;
	cout<<"Result of multiplication: ";	a.display(first4);	cout<<endl;
}
```

Doubly Linked List

```
#include<iostream.h>
#include<conio.h>
#include<process.h>
class list
{
	int data;
	list *next;
	list *prev;
 public:
	void create();
	void insBefore();
	void insAfter();
	void delSpecific();
	void traverse();
	void reverse();
};
list *first=NULL;
void list::create()
{
	list *temp;
	temp=new list;
	cout<<"\nent er the data:";
	cin>>temp->data;
```

```
	temp->prev=NULL;
	temp->next=NULL;
	if(first==NULL)
		first=temp;
	else
	{
		for(list *curr=first;curr->next!=NULL;curr=curr->next);
		curr->next=temp;
		temp->prev=curr;
	}
	traverse();
}
void list::insBefore()
{
	int x,flag=0;
	list *temp;
	temp=new list;
	cout<<"\nenter the element before which the node is to be inserted:";
	cin>>x;
	cout<<"\nenter the data for the node:";
	cin>>temp->data;
	temp->prev=temp->next=NULL;
	if(first==NULL)
		cout<<"\nlink is empty";
	else if(x==first->data)
		{
			temp->next=first;
			first=temp;
			flag=1;
		}
		else
		{

			for(list *curr=first;curr!=NULL;curr=curr->next)
			{
				if(curr->data==x)
				{
				temp->next=curr;
				temp->prev=curr->prev;
				curr->prev->next=temp;
				curr->prev=temp;
				flag=1;
				break;
				}
			}

		}
		if(flag==0) cout<<"\nnode not found";
		traverse();
}
void list::insAfter()
{
```

```
	int x,flag=0;
	list *temp;
	temp=new list;
	cout<<"\nenter the element after which the node is to be inserted:";
	cin>>x;
	cout<<"\nenter the data for the node:";
	cin>>temp->data;
	temp->prev=temp->next=NULL;
	if(first==NULL)
		cout<<"\nlink is empty";
	else if(first->data!=x)
	{
		temp->next=first;
		first->prev=temp;
		first=temp;
	}
	else
	{
		for(list *curr=first;curr!=NULL;curr=curr->next){
			if(curr->data==x&&curr->next!=NULL)
			{
				temp->next=curr->next;
				curr->next->prev=temp;
				curr->next=temp;
				temp->prev=curr;
				flag=1;
				break;
			}
			else if(curr->data==x&&curr->next==NULL)
			{
				curr->next=temp;
				temp->prev=curr;
				flag=1;
				break;
			}

		}
	}
	if(flag==0)		cout<<"\nnode not found\n";
	traverse();
}
void list::delSpecific()
{
	int x,flag=0;
	cout<<"\nenter the element to be deleted:";
	cin>>x;

	if(first==NULL)
		cout<<"\nlist is empty";
	else if(first->data==x)
		{
			list *temp=first;
```

```
                first=first->next;
                first->prev=NULL;
                delete temp;
                flag=1;
        }
        else
        {
                for(list *curr=first;curr!=NULL;curr=curr->next){
                if(curr->data==x&&curr->next!=NULL)
                {
                        curr->prev->next=curr->next;
                        curr->next->prev=curr->prev;
                        delete curr;
                        flag=1;
                        break;
                }
                else if(curr->data==x)
                {
                        curr->prev->next=NULL;
                        delete curr;
                        flag=1;
                        break;
                }
                }

        }
if(flag==0) cout<<"\nnode not found";
        traverse();
}
void list::traverse()
{
        if(first==NULL)
                cout<<"\nlist is empty";
        else
        {
                cout<<"\nthe list is:\n";
                for(list *curr=first;curr!=NULL;curr=curr->next)
                        cout<<curr->data<<"\t";
        }
}
void list::reverse()
{
        list *temp1,*temp2;
        temp1=new list;
        temp2=new list;
        temp1=temp2=NULL;
        while(first!=NULL)
        {
                temp1=temp2;
                temp2=first;
                first=first->next;
                temp2->next=temp1;
```

```
                temp1->prev=temp2;
        }
        first=temp2;
        cout<<"\nreversed node is:";
        traverse();
}
void main()
{
        clrscr();
        list l;
        int ch;
        do
        {
                cout<<"\n1 create\n2 insert before\n3 insert after\n4 delete
specific\n5traverse\n6.reverse\n0.exit";
                cout<<"\nenter your choice:";
                cin>>ch;
                switch(ch)
                {
                        case 1:l.create();break;
                        case 2:l.insBefore();break;
                        case 3:l.insAfter();break;
                        case 4:l.delSpecific();break;
                        case 5:l.traverse();break;
                        case 6:l.reverse();break;
                        case 0:break;
                        default:cout<<"\ninvalid input";break;
                }
        }while(ch!=0);
        getch();
}
```

Tree Data Stucture

```
#include<iostream.h>
#include<conio.h>
#include<process.h>
#include<stdio.h>
#include<string.h>

class treenode
{
        treenode* leftchild;
        int data;
        treenode* rightchild;
        public:
        treenode(int element=0)
        {
                data=element;
                leftchild=NULL;
                rightchild=NULL;
        }
        treenode* insert(int,treenode*);
```

```
        void inorder(treenode*);
        void preorder(treenode*);
        void Iterpreorder(treenode*);
 //     void postorder(treenode*);
        treenode* parent(treenode *curr,int ele, treenode *prev);
        void level_order(treenode*);
        treenode *copy(treenode*);
        int treenode::depth(treenode *ptr);
        void treenode::ancestors(int ele);
};


treenode* root=NULL;

void treenode::ancestors(int ele)
{
        treenode *p;
        p=parent(root, ele, root);
        while(1)
        {
                p=parent(root, p->data, root);
                if(p==root)     break;
        }
}


treenode* treenode::parent(treenode *curr,int ele, treenode *prev)
{
        if(curr!=NULL)
        {
                parent(curr->leftchild, ele, curr);
                if(ele==curr->data)
                {
                        cout<<"\n parent : "<<prev->data;
                        return prev;
                }
                parent(curr->rightchild, ele , curr);
        }
}

int treenode::depth(treenode *ptr)
{
int ldepth,rdepth;
if(ptr==NULL)           return 0;
else
   {   ldepth=depth(ptr->leftchild);
        rdepth=depth(ptr->rightchild);
        if(ldepth>rdepth)       return ldepth+1;
        else                    return rdepth+1;
   }
}
```

```
/*
void treenode::compare(treenode *ptr,treenode *temp,int *f)
{

   if(ptr)
   {
        if(ptr->leftchild)
                return 0;
        if(ptr->rightchild) if(ptr->rightchild!=temp->rightchild) return 0;
     return 1;
   }

}
*/

treenode *treenode::copy(treenode *ptr)
{
   treenode *temp;
   if(ptr)
   {
        temp=new treenode;
        if(ptr->leftchild) temp->leftchild=copy(ptr->leftchild);
        if(ptr->rightchild) temp->rightchild=copy(ptr->rightchild);
        temp->data=ptr->data;
        return(temp);
   }
   return(NULL);
}


/*
void treenode::IterPreorder(treenode *root)
{
        if (root==NULL)      return;
        int top=-1;   treenode *s[10],*ptr;              //create an empty stack
        if(root)  s[++top]=root;                         //place the root into the stack
        while(top>=0) {
                ptr=s[top--];
                cout<<ptr->data;
                s[++top]=ptr->rightchild;
                s[++top]=ptr->leftchild;
        }
}
*/
treenode *treenode::insert(int item,treenode* root)
{
        treenode* temp=new treenode(item);
        if(root==NULL)
        {
                root=temp;
                return root;
```

```
        }
        else
        {
                char direction[20];
                cout<<"enter direction in uppercase: ";
                cin>>direction;
                treenode* current;
                treenode* prev;
                prev=NULL;
                current=root;
                int i;
                for(i=0;i<strlen(direction)&&current!=NULL;i++)
                {
                        prev=current;
                        if(direction[i]=='L')
                                current=current->leftchild;
                        else
                                current=current->rightchild;
                }
                if(current!=NULL||i!=strlen(direction))
                {
                        cout<<"insertion not possible";
                        delete temp;
                        return root;
                }
                if(direction[i-1]=='L')
                        prev->leftchild=temp;
                else
                        prev->rightchild=temp          ;
        }
        return root;
}
void treenode::preorder(treenode* ptr)
{
int top=-1;
treenode *stack[10];

do{
   for(;ptr;ptr=ptr->leftchild)
   {
        cout<<ptr->data<<" ";
        stack[++top]=ptr;
   }


   if(top>=0)        ptr=stack[top--];
   else             break;

   ptr=ptr->rightchild;

}while(1);
}
```

```
void treenode::inorder(treenode* ptr)
{
int top=-1;
treenode *stack[10];

do{
   for(;ptr;ptr=ptr->leftchild)
         stack[++top]=ptr;

   if(top>=0)        ptr=stack[top--];
   else              break;

   cout<<ptr->data;
   ptr=ptr->rightchild;

}while(1);
}

/*
void treenode::postorder(treenode* current)
{
if(current)
{
postorder(current->leftchild);
postorder(current->rightchild);
cout<<current->data<<"  ";
}
}

*/
void treenode::level_order(treenode* ptr)
{
   int front=-1;
   int rear=-1;
   treenode *Q[10];
   if(!ptr)   return;
   Q[++rear]=ptr;
   do
   {
         ptr=Q[++front];
      //   if(ptr)
//         {
            cout<<ptr->data<<" ";
             if(ptr->leftchild)      Q[++rear]=ptr->leftchild;
             if(ptr->rightchild)     Q[++rear]=ptr->rightchild;
  //       }

   }while(front!=rear);
}

void main()
{
```

```
        clrscr();
        int ch,f,ele;
        int a;
        treenode x,*p;

        //treenode *p=NULL;
        //treenode* ne;
do
{
cout<<"\n1:create,2:pre.3:in,4:post,5:parent 6.depth 7.copy 8.ancestors 9.exit\n";
cin>>ch;
switch(ch)
{
        case 1: cout<<"enter element";
                cin>>a;
                root=x.insert(a,root);
                break;
        case 2:
                x.preorder(root);
                break;
        case 3: x.inorder(root);
                break;
//      case 4: x.postorder(root);
  //            break;
        case 5:
                cout<<"Enter the ele: ";
                cin>>ele;
                x.parent(root,ele,root);
                break;
//      case 5: x.level_order(root);
  //            break;
        case 6: cout<<"\nDepth of the tree: "<<x.depth(root)<<endl;
                break;
        case 7: treenode *ne=x.copy(root);
                cout<<" ";
                x.preorder(ne);
                break;
        case 8:
                cout<<"Enter the ele: ";
                cin>>ele;
                x.ancestors(ele);
                break;

        case 9: exit(0);
}
}while(1);
getch();
}
```

Postfix Expression Tree

```
#include<iostream.h>
#include<string.h>
#include<conio.h>
```

```
#include<ctype.h>

class exp_tree
{
   public:
   exp_tree *left;
   char data;
   exp_tree *right;

  exp_tree(){ }

            exp_tree(char ele)
            {
                  data=ele;
                  left=NULL;
                  right=NULL;
            }
            exp_tree* create(char);
            void inorder(exp_tree *);
};

exp_tree *root=NULL;

exp_tree* exp_tree::create(char ele)
{
   exp_tree *temp=new exp_tree(ele);
   return temp;
}

void exp_tree::inorder(exp_tree *ptr)
{
   if(ptr!=NULL)
         {
         inorder(ptr->left);
         cout<<ptr->data<<"   ";
         inorder(ptr->right);
         }
}


int main()
{
   char postfix[10];
   int i=0,top=-1;
   clrscr();
   exp_tree *stack[10],obj;
   cout<<"Enter the postfix expression: "; cin>>postfix;

   while(i<strlen(postfix))
   {
         if(isalpha(postfix[i]))
                  stack[++top]=obj.create(postfix[i]);
```

```
        else
        {
         root=obj.create(postfix[i]);
        root->right=stack[top--];
        root->left=stack[top--];
        stack[++top]=root;
      }
      i++;
   }
obj.inorder(root);
getch();
}
```

Binary Search Tree

```
#include<iostream.h>
#include<process.h>
#include<conio.h>
class bst
{
        bst *lchild;
        int data;
        bst *rchild;
public:
        bst(int ele=0)
        {
                lchild=NULL;
                rchild=NULL;
                data=ele;
        }

        void insert(int);
        void search(int);
        void display(bst *root);
        void del(int);
};

bst *root=NULL;

void bst::insert(int ele)
{
        bst *temp=new bst(ele);
        if(root==NULL){root=temp;return;}

        else
        {
                bst *curr=root,*prev=NULL;
                while(curr)
                {
                        prev=curr;
                        if(temp->data<curr->data)     curr=curr->lchild;
                        else if(temp->data>curr->data)          curr=curr->rchild;
                        else
```

```
                        {
                                cout<<"Insertion is not possible";
                                return;
                        }
                }

                if(temp->data>prev->data)
                        prev->rchild=temp;
                else if(temp->data<prev->data)
                        prev->lchild=temp;
        }
        return;
}

void bst::display(bst *ptr)
{
        if(ptr){
                display(ptr->lchild);
                cout<<" "<<ptr->data;
                display(ptr->rchild);
        }
}

bst* bst::del()
{
bst *c, *p, *s,*q;
cout<<"Enter the key value to
be deleted ";
cin>>ele;
if(root==NULL)
return root;
p=NULL;
c=root;
while(c!=NULL)
{
if(ele==c->data)
break;
p=c;
if(ele<c->data)
c=c->lchild;
else
c=c->rchild;
}
if(c==NULL)
return root;
if(c->lchild==NULL)
q=c->rchild;
else if(c->rchild==NULL)
q=c->lchild;
else
{
```

```
s=c->rchild;
while(s->lchild)
s=s->lchild;
s->lchild=c->lchild;
q=c->rchild;
}
if(!p)
{
delete c;
return q;
}
if(c==p->lchild)
p->lchild=q;
else
p->rchild=q;
delete(c);
return(root);
}


void bst::search(int ele)
{

        if(root==NULL)
        {
                cout<<"tree is empty";
                return;
        }
        bst *curr=root;
        while(curr!=NULL)
        {
                if(curr->data==ele)
                {
                        cout<<"Element found";
                        return;
                }
                else if(curr->data>ele)curr=curr->lchild;
                else                    curr=curr->rchild;
        }
        cout<<"Element not found";
}



void main()
{
        int ch,ele;
        bst b;
        while(1){
        cout<<"\n1. insert 2. display 3. search 4. delete 5. exit\n";
        cin>>ch;
```

```
        switch(ch)
        {
                case 1: cout<<"Enter the element to be inserted: ";
                        cin>>ele;
                        b.insert(ele);
                        b.display(root);
                        break;
                case 2: b.display(root);
                        break;
                case 3: cout<<"Enter the element to be found: ";
                        int ele;
                        cin>>ele;
                        b.search(ele);
                        break;
                case 4: cout<<"Enter the element to be deleted: ";
                        //int ele;
                        cin>>ele;
                        b.del(ele);
                        break;
                case 5: exit(0);
        }
        }
}
```

HEAP

```
#include<iostream.h>
#include<conio.h>


class maxheap
{
        int key;
public: void insert();
        void sort();
        void adjust(maxheap*,int);
        void display();
   void delMax();
};

maxheap h[50];

int n=0;
void maxheap::insert()
{
        if(n==50)
                cout<<"\nHeap Is Full";
        else
        {
                maxheap temp;
                cout<<"\nEnter the element:";
                cin>>temp.key;
```

```
                int i=++n;
                while(i!=1&&temp.key>h[i/2].key)
                {
                        h[i]=h[i/2];
                        i=i/2;
                }
                h[i]=temp;
        }
        display();
}
void maxheap::sort()
{
        int m,temp;
        for(m=n;m>1;m--)
        {
//          display();
            temp=h[1].key;
            h[1].key=h[m].key;
            h[m].key=temp;
            adjust(h,m-1);
                display();
        }
}


void maxheap::adjust(maxheap h[],int no)
{
   int j=1,temp,t;
   temp=h[j].key;

        while((j*2)<=no&& (j*2+1)<=no&&(h[j*2].key>temp||h[j*2+1].key>temp))
        {

           if(h[j*2].key>h[j*2+1].key)
           {
             h[j].key=h[j*2].key;
             h[j*2].key=temp;
             j=j*2;
           }
           else if(h[j*2].key<h[j*2+1].key)
           {
                h[j].key=h[j*2+1].key;
                h[j*2+1].key=temp;
                j=j*2+1;
             }

        }

        if((j*2)<=no)
        {
           if(h[j].key<h[2*j].key)
           {
```

```
                t=h[j].key;
                h[j].key=h[2*j].key;
                h[2*j].key=t;
            }
        }

}

void maxheap::display()
{   cout<<"\nHeap Is: \n";
   for(int i=1;i<=n;i++)
        cout<<h[i].key<<"\t";
}
void maxheap::delMax()
{
        int temp;
   if(n==0)
        {
            cout<<"Heap is empty";
            return;
        }

        cout<<"Deleted item is"<<h[1].key;

   h[1].key=h[n].key;
   n--;

   int i=1;

   while(((2*i)<=n && (2*i+1)<=n) && (h[i].key<h[2*i].key || h[i].key<h[2*i+1].key))
      if(h[2*i].key>=h[2*i+1].key)
            {
               temp=h[i].key;
                    h[i].key=h[2*i].key;
                    h[2*i].key=temp;
                    i=i*2;
            }
            else
            {
        temp=h[i].key;
                    h[i].key=h[2*i+1].key;
                    h[2*i+1].key=temp;
                    i=2*i+1;
            }

   if(((2*i)<=n) && h[i].key<h[2*i].key)
   {
      temp=h[i].key;
                    h[i].key=h[2*i].key;
                    h[2*i].key=temp;
      i=i*2;
   }
```

```
   if(((2*i+1)<=n) && h[i].key<h[2*i+1].key)
   {
      temp=h[i].key;
                        h[i].key=h[2*i+1].key;
                        h[2*i+1].key=temp;

      i=i*2+1;
   }
}

int main()
{
        maxheap m;
        int ch;
        do
        {
                cout<<"\n1. Insert\n2. Sort\n3. Display\n4. delete\n0. Exit";
                cout<<"\n Enter Your Choice";
                cin>>ch;
                switch(ch)
                {
                        case 1: m.insert();break;
                        case 2: m.sort();break;
                        case 3: m.display();break;
                        case 4: m.delMax();break;
                        case 0: break;
                        default: cout<<"\nInvalid choice";break;
                }
        }while(ch!=0);
        return 1;
}
```

Threaded Binary Tree

```
        #include<iostream.h>
        #include<conio.h>
        #include<process.h>
        enum boolean{false,true};

        class thread
        {
                enum boolean left;
                thread *lchild;
                int data;
                thread *rchild;
                enum boolean right;
                public:
                        void insert(int num);
                        void inorder();
                        void inorder_new();
                        void search(int);
        };
```

```
thread *head=NULL;
void thread::insert(int num)
{
	if(head==NULL)
	{
		head=new thread;
		head->left=false;
		head->data=num;
		head->lchild=NULL;
		head->rchild=NULL;
		head->right=false;
		return;
	}
	thread *temp=new thread;
	temp->left=true;
	temp->data=num;
	temp->right=true;

	if(head->lchild==NULL)
	{
		head->lchild=temp;
		temp->lchild=NULL;
		temp->rchild=head;
		return;
	}
	else
	{
		if(head->rchild==NULL)
		{
			head->rchild=temp;
			temp->lchild=head;
			temp->rchild=NULL;
			return;
		}
	}

	thread *ptr=head;
	while(ptr->right!=true)
		ptr=ptr->rchild;

	if(ptr->left==true)
	{
		temp->lchild=ptr->lchild;
		temp->rchild=ptr;
		ptr->lchild=temp;
		ptr->left=false;
	}
	else
	{
		temp->lchild=ptr;
		temp->rchild=NULL;
		ptr->rchild=temp;
```

```
                ptr->right=false;
        }
}

void thread::inorder()
{
        thread *p=head;
        if(head->rchild==NULL&&head->lchild==NULL)
        {
                cout<<head->data<<" ";
                return;
        }

        while(p)
        {
                while(p->left!=true)    p=p->lchild;

                cout<<p->data<<" ";
                while(p->right==true)
                {
                        p=p->rchild;
                        if(p==NULL)  break;
                        cout<<p->data<<" ";
                }

                if(p!=NULL)   p=p->rchild;
        }
}

void thread::inorder_new()
{
        thread *p=head->lchild;
        while(p)
        {
                while(p->left!=true)    p=p->lchild;
                cout<<p->data<<" ";
                while(p->right==true)
                {
                        p=p->rchild;
                        cout<<p->data<<" ";
                }
                p=p->rchild;
        }
}


void main()
{
        int ch,ele;
        thread t;
        clrscr();
        while(1)
```

```
        {
                clrscr();
                cout<<"1. insert 2. inorder 3. search 4. exit\nEnter your choice: \n";
                cin>>ch;

                switch(ch)
                {
                        case 1: cout<<"Enter the element to be inserted: ";
                                cin>>ele;
                                t.insert(ele);
                                t.inorder_new();
                                getch();
                                break;
                        case 2: t.inorder();getch();break;

                        case 4: exit(0);break;
                }
        }
}
```

DFS and BFS

```
#include<iostream>
using namespace std;

void bfs(int a[20][20],int n,int source);
void dfs(int a[20][20],int n,int source);

int main()
{
   int a[20][20],source, n;
   int i,j;

   cout<<"Enter the no of vertices: ";
   cin>>n;

   cout<<"Enter the adjacency matrix: ";
   for(i=1;i<=n;i++)
      for(j=1;j<=n;j++)
         cin>>a[i][j];

   cout<<"Enter the source: ";
   cin>>source;
   cout<<"\n BFS: ";
   bfs(a,n,source);
   cout<<"\n DFS: ";
   dfs(a,n,source);
   return 1;
}

//Algorithm BFS
// Mark all the n vertices as not visited.
// insert source into Q and mark it visited
```

```
//while(Q is not empty)
//{
// delete Q element into variable u
//  place all the adjacent (not visited) vertices of u into Q and also mark them visited
// print u
//}

void bfs(int a[20][20],int n,int source)
{
   int visited[10],u,v,i;

   for(i=1;i<=n;i++)
      visited[i]=0;

   int Q[20],f=-1,r=-1;

   Q[++r]=source;
   visited[source]=1;

   while(f<r)
   {
      u=Q[++f];
      for(v=1;v<=n;v++)
      {
         if(a[u][v]==1 && visited[v]==0)
         {
            visited[v]=1;
            Q[++r]=v;

         }
      }
      cout<<u<<" ";
   }
}

//Algorithm DFS
// Mark all the n vertices as not visited.
// insert source into stack and mark it visited
//while(Stack is not empty)
//{
// delete Stack element into variable u
//  place all the adjacent (not visited) vertices of u into Stack and also mark them visited
//}
// print u

void dfs(int a[20][20],int n,int source)
{
   int visited[10],u,v,i;

   for(i=1;i<=n;i++)
      visited[i]=0;
```

```
   int S[20],top=-1;

   S[++top]=source;
   visited[source]=1;

   while(top>=0)
   {
      u=S[top--];
      for(v=1;v<=n;v++)
      {
         if(a[u][v]==1 && visited[v]==0)
         {
            visited[v]=1;
            S[++top]=v;
         }
      }
      cout<<u<<" ";
   }
}
```

Merge Sort

```
#include <iostream.h>
#include <conio.h>

void merge(int a[],int, int , int );

void mergesort(int a[], int low, int high)
{
   int mid;
   if (low < high)
   {
      mid=(low+high)/2;
      mergesort(a,low,mid);
      mergesort(a,mid+1,high);
      merge(a,low,high,mid);
   }
   return;
}
void merge(int a[], int low, int high, int mid)
{
   int i, j, k, c[50];
   i = low;
   k = low;
   j = mid + 1;
   while (i<=mid&&j<=high)
   {
      if (a[i] < a[j])      c[k++] = a[i++];
      else                        c[k++] = a[j++];

   }
   while (i <= mid)       c[k++] = a[i++];
   while (j <= high)      c[k++] = a[j++];
```

```
   for (i = low; i < k; i++) a[i] = c[i];

}
int main()
{
        int a[20], i, b[20],n;
        cout<<"Enter the value of n: "; cin>>n;
   cout<<"enter  the elements\n";
   for (i=0;i<n; i++) cin>>a[i];
   mergesort(a, 0, n-1);
   cout<<"sorted array\n";
   for (i = 0; i < n; i++) cout<<a[i]<<"  ";

   getch();
}
```